Apple Assembly Line

$1.80

Volume 6 -- Issue 7 April, 1986

In This Issue...

65816 Books

The race is on! "Programming the 65816", by David Eyes from Prentice-Hall, originally scheduled for publication last October, is now expected in late April. "65816/65802 Assembly Language Programming", by Michael Fischer from Osborne/McGraw-Hill, scheduled for May publication, is now also due in late April. We have plenty of copies of these books on order, and a long list of patient people waiting for complete information on programming these powerful new chips. Coming Soon...

More Memory Expansion

We'd like to call your attention to the new ad for Applied Engineering's RamFactor board. This is a "Slinky" style memory expansion card for any standard slot of an Apple II, II+, or //e. We've been doing some of the firmware for this product, and it's been a delight to work with.

One thing the ad doesn't really emphasize is the power and flexibility of the program switcher firmware. You can set up the card with a variable number of variable-sized partitions and then switch between them almost instantly. Any partition can be based on any operating system, or on your own program. Couple this with the battery backup option (it's really more of an uninterruptible power suppply for the card) and you have what amounts to a hard disk operating at RAM speed!

Tools for Restoring Lost Catalogs........Bob Sander-Cederlof

From time to time it happens. One way or another I manage to clobber a catalog track on a disk. I have done it three times to Volume 1 in the DOS partition on my 10-megabyte Sider. (All it takes is "INIT HELLO,V1", forgetting that the last slot I accessed was the Sider's.)

All of the other tracks are still intact, but there is no way to get to them because the catalog is totally wiped out. One solution would be to have an accurate backup floppy for each Sider volume. This should be especially easy for Volume 1, because it is mostly standard Sider utilities. Mostly.... I have modifed several of them, and somehow I almost always have several programs-under-development that end up in V1. Of course, I could just as easily destroy the catalog track on any other volume, or any floppy for that matter.

It is for mistakes like mine that the program FIXCAT in "Bag of Tricks" was invented. FIXCAT looks over a diskette, finds all the sectors which look like they contain track/sector lists, and tries to piece together a new catalog track. Even though it is fairly automatic, I find it very difficult to use. I am always getting confused between old (deleted) copies of files and the current ones, and my disks usually have at least 2 or 3 dozen active files.

Recently it happened again. In fact, while I was working on one of the other articles in this issue of AAL. I decided to write a couple of utilities to help me make more effective use of FIXCAT. My new tools turn out to be useful even without FIXCAT, and you might enjoy just playing with them.

I assume you have a copy of "Beneath Apple DOS", or some other reference work which explains the format of DOS disks, catalog tracks, and track/sector lists.

The first tool I wrote looks through the tracks and sectors of a damaged disk for any sectors containing what could be track/sector lists. When one is found, I display the location of the supposed TS-list, all of the track/sectors in the list, and the first 64 bytes of the first data sector of the supposed file. Here is an example of the display:

```
  03-5:  03-4 03-3 03-2
         07 02 09 E8 03 81 2E 4C 49 46 00 16 F2 03 2A C0 ...h...LIF..r.*@
         06 08 53 41 56 45 81 42 43 44 81 4D 41 47 49 43 ..SAVE.BCD.MAGIC
         00 08 FC 03 2A C0 20 2D 00 05 06 04 54 00 0B 10 ..|.*@ -....T...
         04 87 4C 44 41 81 23 30 00 0C 1A 04 2E 31 85 53 ..LDA.#0.....1.S
```

The first 64 bytes are displayed in both hexadecimal and in ASCII, with periods being substituted for unprintable characters.

Having this information on paper before starting up FIXCAT is a big help. I can peacefully analyze the data at my desk, without the fear and panic associated with making "life and death" decisions at the keyboard. The first few bytes of a

S-C Macro Assembler Version 2.0......DOS $100, ProDOS $100, both for $120
Version 2.0 DOS Upgrade Kit for 1.0/1.1/1.2 owners....................$20
ProDOS Upgrade Kit for Version 2.0 DOS owners.........................$30
Source Code of S-C Macro 2.0 (DOS only)..................additional $100
Full Screen Editor for S-C Macro (with complete source code)..........$49
S-C Cross Reference Utility......without source code $20, with source $50
RAK-Ware DISASM...$30
Source Code for DISASM....................................additional $30
S-C Word Processor (with complete source code)........................$50
DP18 Source and Object..$50
Double Precision Floating Point for Applesoft (with source code)......$50
ES-CAPE (Extended S-C Applesoft Program Editor).........
Including Version 2.0 With Source Code..........$50
ES-CAPE Version 2.0 and Source Code Update (for Registered Owners)....$30
"Bag of Tricks", Worth & Lechner, with diskette.............($39.95) $36 *
"Bag of Tricks 2"...($49.95) $45 *
MacASM -- Macro Assembler for MacIntosh (Mainstay).........($150.00) $100 *
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk...............................$15
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each
(Available: 6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048,
8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks.........................each $15, or any four for $45
Each disk contains the source code from three issues of AAL,
saving you lots of typing and testing.
The quarters are Jan-Mar, Apr-Jun, Jul-Sep, and Oct-Dec.
(All source code is formatted for S-C Macro Assembler. Other assemblers
require some effort to convert file type and edit directives.)

Diskettes (with hub rings)......................... package of 20 for $20 *
Vinyl disk pages, 6"x8.5", hold two disks each..................10 for $6 *
Diskette Mailing Protectors (hold 1 or 2 disks).............40 cents each
(Cardboard folders designed to fit 6"X9" Envelopes.) or $25 per 100 *
Envelopes for Diskette Mailers............................ 6 cents each

65802 Microprocessor (Western Design Center)..................($95) $50 *
quikLoader EPROM System (SCRG)...............................($179) $170 *
PROmGRAMER (SCRG)..($149.50) $140 *
Switch-a-Slot (SCRG).......................................($179.50) $170 *
Extend-a-Slot (SCRG)..($35) $32 *

"65816/65802 Assembly Language Programming", Fischer........($19.95) $18 *
"Programming the 65816", Eyes...............................($22.95) $21 *
"Apple //e Reference Manual", Apple Computer................($24.95) $23 *
"Apple //c Reference Manual", Apple Computer................($24.95) $23 *
"ProDOS Technical Reference Manual", Apple Computer.........($29.95) $27 *
"Now That You Know Apple Assembly Language...", Gilder......($19.95) $18 *
"Apple ProDOS: Advanced Features for Programmers", Little..($17.95) $17 *
"Inside the Apple //c", Little..............................($19.95) $18 *
"Inside the Apple //e", Little..............................($19.95) $18 *
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95) $18 *
"Apple][Circuit Description", Gayler......................($22.95) $21 *
"Understanding the Apple II", Sather........................($22.95) $21 *
"Understanding the Apple //e", Sather.......................($24.95) $23 *
"Enhancing Your Apple II, vol. 1", Lancaster................($15.95) $15 *
"Enhancing Your Apple II, vol. 2", Lancaster................($17.95) $17 *
"Assembly Cookbook for the Apple II/IIe", Lancaster.........($21.95) $20 *
"Beneath Apple DOS", Worth & Lechner........................($19.95) $18 *
"Beneath Apple ProDOS", Worth & Lechner.....................($19.95) $18 *
"Real Time Programming -- Neglected Topics", Foster..........($9.95) $9 *
"Microcomputer Graphics", Myers.............................($14.95) $14 *
"Assem. Language for Applesoft Programmers", Finley & Myers.($18.95) $18 *
"Assembly Lines -- the Book", Wagner........................($19.95) $18 *
"AppleVisions", Bishop & Grossberger........................($39.95) $36 *

* On these items add $2.00 for the first item and
$.75 for each additional item for US shipping.
Foreign customers inquire for postage needed.
Texas residents please add 6 1/8 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** Master Card, VISA, Discover and American Express ***

file will usually reveal what type of file it is.

If it is a source code file for the S-C Macro Assembler, Integer BASIC, or Applesoft, it will begin with a two-byte length for the file. Binary files begin with the load address, then the length. Text files start right in with data in ASCII, normally with all the high bits on. Since I almost always have a line near the beginning of my source files which contains the file name, I can usually read that file name in the dump of the first 64 bytes.

The FIND.TS.LISTS program is fairly short and simple. Starting from the bottom, the subroutine READTS at lines 2370-2430 calls on RWTS to read a particular track and sector. I elected to use my own IOB, rather than the one inside DOS at $B7E8. For simplicity's sake I assembled in the slot, drive, and volume information in my IOB. READTS only has to store the desired track and sector numbers. and call RWTS. I limited error handling to just re-calling RWTS, in the hopes of eventually succeeding. Should this begin to be a problem, I could print out an error message and either quit or continue with the next sector.

The subroutine READ.NEXT.SECTOR, lines 2200-2350, is used to scan through the disk from beginning to end. TS-lists cannot be in track 0, so I start with track 1. Since DOS allocates sectors in a track starting with sector $0F and going backwards to sector $00, I decided to scan the same way. This makes the files found list more closely to the same order as they were in the original catalog. I first advance the track/sector to the next one, then read it. Thus after reading, CUR.TRACK and CUR.SECTOR are pointing to the one we just read.

Now back to the top. Lines 1100-1130 start CUR.TRACK and CUR.SECTOR at 0. The first call to READ.NEXT.SECTOR will advance them to track 1, sector $0F. Successive calls will read the rest of track 1, then advance to track 2, and so on until we have finished track $22. When we try to read track $23, which does not exist, READ.NEXT.SECTOR will return with carry set and our program will end.

Lines 1170-1290 examine the data in the sector just read to see if it might be a track/sector list. The method I use is to require that there be at least one TS-pair, at BUF+12. I also require that all of the bytes beyond BUF+12 are within the range of valid track-sector pairs. If any bytes are out of range, I assume the current sector is not a TS-list. My tests seem to be adequate, because with every disk I have used it on it found all and only the TS-lists.

Having found TS-list, I call DISPLAY.TS.LIST to display it. Lines 1450-1540 display the location of the TS-list. The subroutine PR.TS prints the track and sector numbers from the A- and X-registers in the form "TT-S". Lines 1550-1720 list the TS-pairs in the TS-list, stopping at the first pair with a track number of zero. Up to 8 pairs are listed on a line.

Lines 1330-1430 read the first data sector of the supposed file, and display the first 64 bytes in hex and ASCII. This display is done by calling DISPLAY.NEXT.16 four times.

As it happens, I did have a fairly recently made backup of the clobbered disk. I thought I should also run my program against this good disk, and comparing the two displays would enable me to pinpoint each active file. However, what I really want from the GOOD disk is the information in the CATALOG. I decided to modify FIND.TS.LISTS to be driven from the catalog track, rather than from a search for TS-lists. The result was another useful tool, BIG.CATALOG.DISPLAY.

BIG.CATALOG.DISPLAY has the same kind of output that FIND.TS.LISTS does, except that it also lists the file type, file name, and sector count from the catalog. Information is included for deleted files for which entries are founa in the catalog, as well as all the active files.

The subroutines DISPLAY.TS.LIST, DISPLAY.NEXT.16, SEVEN.SPACES, PR.TS, and READTS are used without any changes from the FIND.TS.LISTS program. Instead of READ.NEXT.SECTOR, I have now READ.NEXT.CATALOG.SECTOR. This starts at track $11, sector $0F, and works back as far as sector $01. A better way might be to follow the actual chain, beginning in the VTOC sector, but the current scheme is easier and works with most of my disks.

Lines 1140-1180 set up the initial catalog track and sector. Lines 1190-1210 read the catalog sector. If the returned status is positive we did read a sector, and continue processing; if not, we are finished. Lines 1220-1250 set up the buffer address in the IOB for reading TS-lists and data sectors: we do not want to read them over the top of the catalog sector we are working with.

Lines 1270-1320 set up a loop for processing each of the seven file entries in the current catalog sector. The "NEXT" part of the loop is at lines 1350-1440. Each catalog entry takes 35 bytes, so lines 1350-1440 add 35 to the pointer.

DISPLAY.DATA.FOR.ONE.FILE first checks for a zero entry, meaning the end of the catalog. A catalog is initialized to all zeroes, so as soon as we find a zero entry we know there are no more files. Next, at lines 1520-1550, I check for a deleted file. If the track number is negative, it is a deleted file. The actual track number of a deleted file is saved on top of the 30th character of the file name, so I pick it up there. Lines 1560-1590 save the track and sector of the TS-list, so I can read it later. Lines 1600-1650 display the file type as a hex value, followed by two dashes.

Lines 1660-1700 print the first 29 characters of the file name. I don't print the last character because for a deleted file it will have been clobbered by saving the track number there. Probably what I should do here is print either the last character for an active file, or some special symbol for a deleted file. You can add that code if you like.

New Debugging Program For S-C Macro Assembler Users

QUICK TRACY is designed to increase your productivity. Below are some of the special features of **QUICK TRACY**:

- Co-resides with the S-C Macro Assembler (version 2.0) to speed up the debugging process, so you don't have to keep on changing from Assembler to debugger and back.
- Has 13 independent conditional breakpoints that are easily set and maintained.
- Works in either 40- or 80-column mode.
- Can automatically output to your printer only the information needed for debugging, or all available information if specified.
- HEX/ASCII dump and disassembly without entering the monitor.
- Realtime execution to help trace through your programs faster.
- Prompting for immediate execution of subroutines.
- Works with graphics programs.

Spend less time debugging and more time programming with QUICK TRACY.

QUICK TRACY costs $35.00.

T'n'T SOFTWARE
624 Charmain Drive
Campbell, CA 95008
(408) 379-0563

We accept Visa or MasterCard.

For Postage and Handling, please include $2.00 if U.S., $5.00 for all other countries. California state residents, please include applicable sales tax.

QUICK TRACY requires S-C Macro Assembler version 2.0, DOS 3.3,
Apple][+ or Apple //e, at least 64K of RAM memory, printer and 1 disk drive.

"Apple" is a registered trademark of Apple Computer, Inc.

Lines 1710-1770 pick up the file size, in number of sectors, and print it as a hex value. The sector count includes the sector for the TS-list.

Lines 1780-1860 read the track/sector list for the file. If either the track number or the sector number is out of range, nothing is read and we skip any further processing for this file.

Lines 1870-1940 read in the first data sector for the file. Again, if either the track or sector number is out of range, we don't try to read it. Finally, lines 1950-2000 display the first 64 bytes of the file.

I hope you find these new tools as useful as I have. Of course, I could hope you will never NEED them, but that would prabably be a vain hope. I also hope you have "Bag of Tricks" or some similar utility to put it all back together after you get the information my tools provide. And if I ever clobber Volume 1 on my Sider again (perish the thought), I intend to modify my copies of DOS so they will not allow me to INIT a volume on the Sider.

```
                  1000 *SAVE S.FIND T/S LISTS
                  1010 *--------------------------------
00-               1020 CUR.SECTOR .EQ 0
01-               1030 CUR.TRACK  .EQ 1
                  1040 *--------------------------------
FDED-             1050 COUT   .EQ $FDED
FD8E-             1060 CROUT  .EQ $FD8E
FDDA-             1070 PRBYTE .EQ $FDDA
03D9-             1080 ENTER.RWTS .EQ $3D9
                  1090 *--------------------------------
                  1100 T
0800- A9 00       1110        LDA #0
0802- 85 00       1120        STA CUR.SECTOR
0804- 85 01       1130        STA CUR.TRACK
0806- 20 D0 08    1140 .1     JSR READ.NEXT.SECTOR
0809- 90 01       1150        BCC .2          GOT A SECTOR, CHECK IT
080B- 60          1160        RTS
                  1170 *---CHECK IF THIS IS T/S LIST----
080C- AD 18 09    1180 .2     LDA BUF+12
080F- F0 F5       1190        BEQ .1          ...NO, TRY NEXT ONE
0811- A0 0C       1200        LDY #12
0813- B9 0C 09    1210 .3     LDA BUF,Y
0816- C9 23       1220        CMP #35
0818- B0 EC       1230        BCS .1          ...NOT VALID TRACK
081A- C8          1240        INY
081B- B9 0C 09    1250        LDA BUF,Y
081E- C9 10       1260        CMP #16
0820- B0 E4       1270        BCS .1          ...NOT VALID SECTOR
0822- C8          1280        INY
0823- D0 EE       1290        BNE .3          ...MORE IN SECTOR TO CHECK
                  1300 *---DISPLAY THE T/S LIST---------
0825- 20 45 08    1310        JSR DISPLAY.TS.LIST
                  1320 *---READ FIRST DATA SECTOR-------
0828- AC 18 09    1330        LDY BUF+12
082B- AE 19 09    1340        LDX BUF+13
082E- 20 E7 08    1350        JSR READTS
                  1360 *---DISPLAY FIRST 64 BYTES-------
0831- A0 00       1370        LDY #0
0833- 20 80 08    1380        JSR DISPLAY.NEXT.16
0836- 20 80 08    1390        JSR DISPLAY.NEXT.16
0839- 20 80 08    1400        JSR DISPLAY.NEXT.16
083C- 20 80 08    1410        JSR DISPLAY.NEXT.16
083F- 20 8E FD    1420        JSR CROUT
0842- 4C 06 08    1430        JMP .1
                  1440 *--------------------------------
```

```
                DISPLAY.TS.LIST
0845- 20 8E FD          JSR CROUT
0848- A5 01             LDA CUR.TRACK
084A- A6 00             LDX CUR.SECTOR
084C- 20 BC 08          JSR PR.TS
084F- A9 BA             LDA #":"
0851- 20 ED FD          JSR COUT
0854- A9 A0             LDA #" "
0856- 20 ED FD          JSR COUT
0859- 20 ED FD          JSR COUT
085C- A0 00             LDY #0
085E- B9 19 09  .1      LDA BUF+13,Y    SECTOR
0861- AA                TAX
0862- B9 18 09          LDA BUF+12,Y    TRACK
0865- F0 18             BEQ .2      ...END OF LIST
0867- 20 BC 08          JSR PR.TS
086A- A9 A0             LDA #" "
086C- 20 ED FD          JSR COUT
086F- 98                TYA
0870- 29 0F             AND #$0F
0872- C9 0E             CMP #$0E
0874- D0 03             BNE .3
0876- 20 AE 08          JSR SEVEN.SPACES
0879- C8        .3      INY
087A- C8                INY
087B- C0 F4             CPY #-12
087D- 90 DF             BCC .1
087F- 60        .2      RTS
                *--------------------------------
                DISPLAY.NEXT.16
0880- 20 AE 08          JSR SEVEN.SPACES
0883- B9 0C 09  .1      LDA BUF,Y
0886- 20 DA FD          JSR PRBYTE
0889- A9 A0             LDA #" "
088B- 20 ED FD          JSR COUT
088E- C8                INY
088F- 98                TYA
0890- 29 0F             AND #$0F
0892- D0 EF             BNE .1
0894- 98                TYA
0895- 38                SEC
0896- E9 10             SBC #16
0898- A8                TAY
0899- B9 0C 09  .2      LDA BUF,Y
089C- 09 80             ORA #$80
089E- C9 A0             CMP #$A0
08A0- B0 02             BCS .3
08A2- A9 AE             LDA #"."
08A4- 20 ED FD  .3      JSR COUT
08A7- C8                INY
08A8- 98                TYA
08A9- 29 0F             AND #$0F
08AB- D0 EC             BNE .2
08AD- 60                RTS
                *--------------------------------
                SEVEN.SPACES
08AE- 20 8E FD          JSR CROUT
08B1- A9 A0             LDA #" "
08B3- A2 07             LDX #7
08B5- 20 ED FD  .4      JSR COUT
08B8- CA                DEX
08B9- D0 FA             BNE .4
08BB- 60                RTS
                *--------------------------------
                PR.TS
08BC- 20 DA FD          JSR PRBYTE
08BF- A9 AD             LDA #"-"
08C1- 20 ED FD          JSR COUT
08C4- 8A                TXA
08C5- 09 B0             ORA #"0"
08C7- C9 BA             CMP #$BA
08C9- 90 02             BCC .1
08CB- 69 06             ADC #6
08CD- 4C ED FD  .1      JMP COUT
                *--------------------------------
```

```
                  2200 *   READ NEXT SECTOR
                  2210 *-------------------------------
                  2220 READ.NEXT.SECTOR
08D0- A6 00       2230        LDX CUR.SECTOR
08D2- A4 01       2240        LDY CUR.TRACK
08D4- CA          2250        DEX           NEXT SECTOR
08D5- 10 07       2260        BPL .1        ...SAME TRACK
08D7- A2 0F       2270        LDX #15       ...NEXT TRACK
08D9- C8          2280        INY
08DA- C0 23       2290        CPY #35
08DC- B0 08       2300        BCS .2        ...END OF DISK
08DE- 84 01       2310 .1     STY CUR.TRACK
08E0- 86 00       2320        STX CUR.SECTOR
08E2- 20 E7 08    2330        JSR READTS
08E5- 18          2340        CLC
08E6- 60          2350 .2     RTS
                  2360 *-------------------------------
08E7- 8E FC 08    2370 READTS STX IOB.SECTOR
08EA- 8C FB 08    2380        STY IOB.TRACK
08ED- A9 08       2390 .2     LDA /IOB
08EF- A0 F7       2400        LDY #IOB
08F1- 20 D9 03    2410        JSR ENTER.RWTS
08F4- B0 F7       2420        BCS .2        ...TRY AGAIN IF ERROR
08F6- 60          2430        RTS
                  2440 *-------------------------------
                  2450 *      IOB FOR RWTS CALLS
                  2460 *-------------------------------
                  2470 IOB
08F7- 01          2480 IOB.TYPE    .HS 01   0--MUST BE $01
08F8- 60          2490 IOB.SLOT16  .HS 60   1--SLOT # TIMES 16
08F9- 01          2500 IOB.DRIVE   .HS 01   2--DRIVE # (1 OR 2)
08FA- 00          2510 IOB.VOLUME  .HS 00   3--DESIRED VOL # (0 MATCHES ANY)
08FB-             2520 IOB.TRACK   .BS 1    4--TRACK # (0 TO 34)
08FC-             2530 IOB.SECTOR  .BS 1    5--SECTOR # (0 TO 15)
08FD- 08 09       2540 IOB.PNTDCT  .DA DCT  6--ADDRESS OF DCT
08FF- 0C 09       2550 IOB.BUFFER  .DA BUF  8--ADDRESS OF DATA
0901- 00 01       2560 IOB.SECTSZ  .DA 256 10--# BYTES IN A SECTOR
0903- 01          2570 IOB.OPCODE  .HS 01  12--0=SEEK, 1=READ, 2=WRITE, OR 4=FORMAT
0904-             2580 IOB.ERROR   .BS 1   13--ERROR CODE: 0, 8, 10, 20, 40, 80
0905-             2590 IOB.ACTVOL  .BS 1   14--ACTUAL VOLUME # FOUND
0906- 60          2600 IOB.PRVSLT  .HS 60  15--PREVIOUS SLOT #
0907- 01          2610 IOB.PRVDRV  .HS 01  16--PREVIOUS DRIVE #
                  2620 *-------------------------------
0908- 00 01 EF
090B- D8          2630 DCT    .HS 0001EFD8
                  2640 *-------------------------------
090C-             2650 BUF    .BS 256
                  2660 *-------------------------------
```

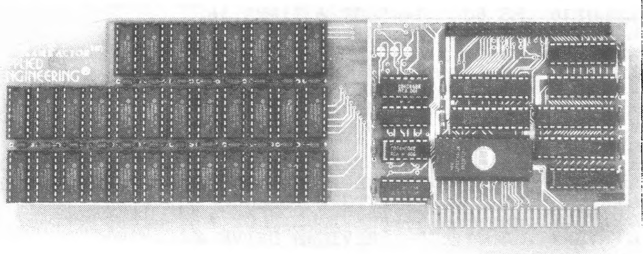
```
                  1000 *SAVE S.BIG CATALOG DISPLAY
                  1010 *-------------------------------
00-               1020 CAT.SECTOR .EQ 0
01-               1030 CAT.TRACK  .EQ 1
02-               1040 CNTR       .EQ 2
03-               1050 PNTR       .EQ 3,4
05-               1060 TS.TRACK   .EQ 5
06-               1070 TS.SECTOR  .EQ 6
                  1080 *-------------------------------
FDED-             1090 COUT   .EQ $FDED
FD8E-             1100 CROUT  .EQ $FD8E
FDDA-             1110 PRBYTE .EQ $FDDA
03D9-             1120 ENTER.RWTS .EQ $3D9
                  1130 *-------------------------------
                  1140 BIG.CATALOG.DISPLAY
0800- A9 0F       1150        LDA #15
0802- 85 00       1160        STA CAT.SECTOR
0804- A9 11       1170        LDA #17
0806- 85 01       1180        STA CAT.TRACK
0808- 20 3C 09    1190 .1     JSR READ.NEXT.CATALOG.SECTOR
080B- 10 01       1200        BPL .2        GOT A SECTOR
080D- 60          1210 .4     RTS
080E- A9 75       1220 .2     LDA #BUF
0810- 8D 68 09    1230        STA IOB.BUFFER
0813- A9 09       1240        LDA /BUF
0815- 8D 69 09    1250        STA IOB.BUFFER+1
```

RamFactor™

All the Performance, Speed, and Software Compatibility of RamWorks™ in a Slot 1 through 7 Card.

That's right! Now Applied Engineering offers you a choice. While RamWorks is the clear winner for the auxiliary slot in a IIe, RamFactor is the standard for slots 1 through 7. Now anyone with an Apple II+, Franklin, or Apple IIe preferring to use slots 1 through 7 can now enjoy the speed and performance that until now was only available with RamWorks.

With RamFactor, you'll be able to instantly add another 256K, 512K, or a full 1 meg on the main board and up to 16 meg with additional piggyback card. And since virtually all software is automatically compatible with RamFactor, you'll immediately be able to load programs into RamFactor for instantaneous access to information. You'll also be able to store more data for larger word processing documents, bigger data bases, and expanded spreadsheets.



Very Compatible

All the leading software is already compatible with RamFactor. Programs like AppleWorks, Pinpoint, BPI, Managing Your Money, Dollars and Sense, SuperCalc 3A, PFS, MouseWrite, MouseDesk, MouseCalc, Sensible Speller, Applewriter IIe, Business Works, ReportWorks, Catalyst 3.0 and more. And RamFactor is fully ProDos, DOS 3.3, Pascal 1.3 and CP/M compatible. In fact, no other memory card (RamWorks excepted) is more compatible with commercial software.

AppleWorks Power

There are other slot 1-7 cards that give AppleWorks a larger desktop, but that's the end of their story. But RamFactor is the only slot 1-7 card that increases AppleWorks internal memory limits, increasing the maximum number of lines permitted in the word processor, and RamFactor is the only standard slot card that will automatically load AppleWorks into RAM dramatically increasing speed and eliminating the time required to access the program disk, it will even display the time and date on the AppleWorks screen with any ProDos clock. RamFactor will automatically segment large files so they can be saved on 5¼", 3½", and hard disks. All this performance is available to anyone with an Apple IIe *or* II+ with an 80 column card. RamFactor, no other standard slot card comes close to enhancing AppleWorks so much.

True 65C816 16 Bit Power

RamFactor has a built-in 65C816 CPU port for direct connection to our IIe 65C816 card for linearly addressing up to 16 meg for the most powerful 16 bit applications. (II+ 65C816 card under development)

Powerful Program Switcher

With RamFactor, you can organize memory into multiple work areas and switch between them. Each work area can contain different programs and even different operating systems. Now you can switch from one program to another or even switch from AppleWorks to DOS 3.3 to CP/M to Pascal to ProDos in under a second. And with our Battery back-up option, you can have permanent storage for up to 20 years.

Quality and Support of the Industry Leader

RamFactor is from Applied Engineering, the largest, most well supported manufacturer of Apple peripherals and the inventor of large RAM cards for the Apple. With our 5 year no hassle warranty and outstanding technical support, you're assured of the most trouble free product you can buy.

Features:

- Up to 16 meg total memory, 256K to 1 meg on main board. Up to 16 meg with additional memory on piggyback card
- Fully Apple II Memory Expansion compatible
- Compatible with Apple IIe, II+ and Franklin
- Battery back-up option allows you to turn on your Apple and run your favorite programs in less than 1 second!
- Automatically recognized by ProDos, DOS 3.3, Pascal and CP/M
- Built-in RamDrive™ software (a true RAM disk not disk caching)
- Systems are directly bootable from RamFactor if desired
- Built-in linear addressing 16 bit co-processor port
- Built-in self diagnostic software
- Automatic expansion with AppleWorks 1.3 or later
- Allows Apple II+ and IIe to run your AppleWorks without buying additional software
- Accelerates AppleWorks
- Displays time and date on the AppleWorks screen with any ProDos clock
- Fits any I/O slot except slot 3
- Fully socketed and user upgradeable
- Much, much more

RamFactor with 256K	**$239**
RamFactor with 512K	**$289**
RamFactor with 1 MEG	**$389**
RamFactor with 2-16 MEG	**CALL**
Battery Back-up Option	**$179**
65C816 16 Bit Card	**$159**

Order RamFactor today . . . with 15 day money back guarantee and our "no hassle" five year warranty. *Call* 9 a.m. to 11 p.m., 7 days, *or send* check or money order to Applied Engineering. MasterCard, Visa and C.O.D. welcome. Texas residents add 5⅛% sales tax. Add $10.00 if outside U.S.A.

Æ Applied Engineering
The Apple enhancement experts.
(214) 241-6060
P.O. Box 798, Carrollton, TX 75006

```
                   1260 *--------------------------------
0818- A9 80        1270        LDA #CAT+11
081A- 85 03        1280        STA PNTR
081C- A9 0A        1290        LDA /CAT+11
081E- 85 04        1300        STA PNTR+1
0820- A9 07        1310        LDA #7
0822- 85 02        1320        STA CNTR
0824- 20 3F 08     1330 .3     JSR DISPLAY.DATA.FOR.ONE.FILE
0827- B0 E4        1340        BCS .4         ...END OF CATALOG
0829- A5 03        1350        LDA PNTR
082B- 69 23        1360        ADC #35
082D- 85 03        1370        STA PNTR
082F- A5 04        1380        LDA PNTR+1
0831- 69 00        1390        ADC #0
0833- 85 04        1400        STA PNTR+1
0835- C6 02        1410        DEC CNTR
0837- D0 EB        1420        BNE .3
0839- 20 8E FD     1430        JSR CROUT
083C- 4C 08 08     1440        JMP .1
                   1450 *--------------------------------
                   1460 DISPLAY.DATA.FOR.ONE.FILE
083F- A0 00        1470        LDY #0
0841- B1 03        1480        LDA (PNTR),Y
0843- D0 02        1490        BNE .1
0845- 38           1500        SEC
0846- 60           1510        RTS
0847- 10 06        1520 .1     BPL .15
0849- A0 20        1530        LDY #32
084B- B1 03        1540        LDA (PNTR),Y       REAL TRACK OF DELETED FILE
084D- A0 00        1550        LDY #0
084F- 85 05        1560 .15    STA TS.TRACK
0851- C8           1570        INY
0852- B1 03        1580        LDA (PNTR),Y
0854- 85 06        1590        STA TS.SECTOR
0856- C8           1600        INY
0857- B1 03        1610        LDA (PNTR),Y       GET FILE TYPE
0859- 20 DA FD     1620        JSR PRBYTE
085C- A9 AD        1630        LDA #"-"
085E- 20 ED FD     1640        JSR COUT
0861- 20 ED FD     1650        JSR COUT
0864- C8           1660 .2     INY
0865- B1 03        1670        LDA (PNTR),Y       PRINT FILE NAME
0867- 20 ED FD     1680        JSR COUT
086A- C0 1F        1690        CPY #31            DON'T PRINT LAST CHAR OF NAME
086C- 90 F6        1700        BCC .2
086E- C8           1710        INY
086F- C8           1720        INY
0870- B1 03        1730        LDA (PNTR),Y
0872- 20 DA FD     1740        JSR PRBYTE
0875- C8           1750        INY
0876- B1 03        1760        LDA (PNTR),Y
0878- 20 DA FD     1770        JSR PRBYTE
                   1780 *---READ T/S LIST----------------
087B- A6 06        1790        LDX TS.SECTOR
087D- E0 10        1800        CPX #16
087F- B0 2B        1810        BCS .9
0881- A4 05        1820        LDY TS.TRACK
0883- C0 23        1830        CPY #35
0885- B0 25        1840        BCS .9
0887- 20 50 09     1850        JSR READTS
088A- 20 B1 08     1860        JSR DISPLAY.TS.LIST
                   1870 *---READ FIRST DATA SECTOR-------
088D- AC 81 09     1880        LDY BUF+12
0890- C0 23        1890        CPY #35
0892- B0 18        1900        BCS .9
0894- AE 82 09     1910        LDX BUF+13
0897- E0 10        1920        CPX #16
0899- B0 11        1930        BCS .9
089B- 20 50 09     1940        JSR READTS
                   1950 *---DISPLAY FIRST 64 BYTES-------
089E- A0 00        1960        LDY #0
08A0- 20 EC 08     1970        JSR DISPLAY.NEXT.16
08A3- 20 EC 08     1980        JSR DISPLAY.NEXT.16
08A6- 20 EC 08     1990        JSR DISPLAY.NEXT.16
08A9- 20 EC 08     2000        JSR DISPLAY.NEXT.16
08AC- 20 8E FD     2010 .9     JSR CROUT
08AF- 18           2020        CLC
08B0- 60           2030        RTS
```

```
                     2040 *--------------------------------
                     2050 DISPLAY.TS.LIST

                     2330 *--------------------------------
                     2340 DISPLAY.NEXT.16

                     2590 *--------------------------------
                     2600 SEVEN.SPACES

                     2680 *--------------------------------
                     2690 PR.TS

                     2790 *--------------------------------
                     2800 *   READ NEXT CATALOG SECTOR
                     2810 *--------------------------------
                     2820 READ.NEXT.CATALOG.SECTOR
093C- A9 75          2830        LDA #CAT
093E- 8D 68 09       2840        STA IOB.BUFFER
0941- A9 0A          2850        LDA /CAT
0943- 8D 69 09       2860        STA IOB.BUFFER+1
0946- A6 00          2870        LDX CAT.SECTOR
0948- A4 01          2880        LDY CAT.TRACK
094A- 20 50 09       2890        JSR READTS
094D- C6 00          2900        DEC CAT.SECTOR
094F- 60             2910        RTS
                     2920 *--------------------------------
0950- 8E 65 09       2930 READTS STX IOB.SECTOR

                     3000 *----------------------------------
                     3010 *      IOB FOR RWTS CALLS
                     3020 *----------------------------------
                     3030 IOB

                     3180 *--------------------------------
0971- 00 01 EF
0974- D8             3190 DCT    .HS 0001EFD8
                     3200 *--------------------------------
0975-                3210 BUF    .BS 256
0A75-                3220 CAT    .BS 256
                     3230 *--------------------------------
```

Writing Messages in Windows..............Bob Sander-Cederlof

The idea for the following program came from some similar code in the Cirtech Flipster software. Their "program manager" software displays a series of messages and menus in selected windows using a simple subroutine.

The windows are not quite as sophisticated as you may be used to if you are a Macintosh fan. This program divides the screen up vertically, with each window running the full screen width. Calls to the program specify which window to write a message into. The JSR MSG.IN.WINDOW is followed by a single byte specifying which window to use, the ASCII text of the message, and a final 00 byte signifying the end of message. MSG.IN.WINDOW first sets up the window, then clears it, then displays the message in it, and then returns to continue execution right after the 00 byte. MSG.IN.WINDOW does not make any provision for saving the previous contents of the screen inside the window and restoring it later. As I said, this is much simpler than Mac windows.

The Apple monitor has built-in window capability, with the current window being defined by four bytes in page zero. $20 is called LEFT, and defines the starting column of a screen line. This is normally 0, meaning the first column. $21 is called WIDTH, and specifies how many characters are in each line. This is usually 40 ($28), but may be 80 ($50) in a //c or enhanced //e in 80-column mode. MSG.IN.WINDOW does not make any changes to LEFT or WIDTH, although you could modify it to do so.

$22 is called WNDTOP, and specifies the top line of the working window. This is usually 0, meaning to start at the top of the screen. It could be as large as 23 ($17), meaning the bottom line of the screen. $23 is called WNDBOT, and specifies the bottom line of the working window. The number in WNDBOT is actually the number of the next line below the working window, and is usually 24 ($18) to specify a window that goes all the way to the bottom of the screen. MSG.IN.WINDOW stores new values in WNDTOP and WNDBOT, according to a table of line numbers called WINDOW.DATA.

My WINDOW.DATA table lists six different windows, but of course you could have as many as you wish. They can even overlap. The table I used contains the line numbers 0, 24, 0, 3, 9, 18, 20, and 24. This corresponds to the following windows:

Index	WNDTOP	WNDBOT	Window	
0	0	24	0-23	<full screen>
1	<better not use!!!>			
2	0	3	0-2	
3	3	9	3-8	
4	9	18	9-17	
5	18	20	18-19	
6	20	24	20-23	

Lines 1080-1130 in the listing below detail the calling sequence for MSG.IN.WINDOW. The test program in lines 1500 and

following shows some actual calls, with a "wait for any keystroke" between messages so you can see it happen.

Lines 1140-1180 save the caller's return address, placed on the stack by the JSR MSG.IN.WINDOW. This address will be used to pick up the calling parameters, and then used to return to the calling program. The subroutine in lines 1400-1460 increments the pointer and picks up the next byte from the calling sequence.

When we are finished displaying the message, the pointer will be pointing at the terminal 00 byte. Placing the pointer address back on the stack lets us use an RTS opcode to return to the caller. This is done in lines 1340-1390.

Lines 1200-1250 pick up the window index from the first byte following the JSR instruction. This indexes the WINDOW.DATA table, so two entries from that table are moved into WNDTOP and WNDBOT. The the monitor HOME subroutine can be called to clear the window and place the cursor in the top-left corner of the window.

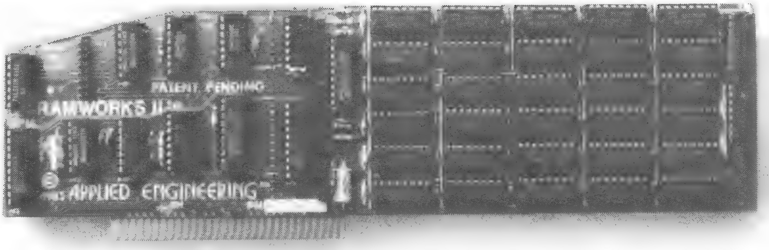
Lines 1270-1330 display the message, if any. If there is no message, there still must be a terminal 00 byte. By judicious use of 8D (return) and 8A (linefeed) characters, you can display the message any way you like. If the message is too large for the window, lines will be scrolled out the top of the window and lost.

The MSG.IN.WINDOW subroutine illustrates a commonly used technique of placing messages to be printed "in-line", like PRINT "message" statements in Applesoft. I personally prefer to collect all my messages together, and use a message number in a register to select which one to print. One problem with my preferred method is that my programs are then easier to disassemble ... if that is a problem. The 6502 was not designed for easy transfer of calling parameters which follow the JSR. (The 65816 makes this kind of code easier, with its stack-relative address mode.)

```
                1000 *SAVE S.MSG INTO WINDOW
                1010 *--------------------------------
FC58-           1020 HOME   .EQ $FC58
FDED-           1030 COUT   .EQ $FDED
                1040 *--------------------------------
00-             1050 PNTR   .EQ $00,01
22-             1060 WNDTOP .EQ $22
23-             1070 WNDBOT .EQ $23
                1080 *--------------------------------
                1090 *   CALL:   JSR MSG.IN.WINDOW
                1100 *           .DA #<window number>
                1110 *           .AS text of message
                1120 *           .HS 00    <end of msg flag>
                1130 *--------------------------------
                1140 MSG.IN.WINDOW
0800- 68        1150        PLA          GET RETURN ADDRESS INTO PNTR
0801- 85 00     1160        STA PNTR     LO BYTE
0803- 68        1170        PLA
0804- 85 01     1180        STA PNTR+1   HI BYTE
                1190 *---SETUP WINDOW TOP & BOTTOM----
0806- 20 2D 08  1200        JSR GET.NEXT.CALL.BYTE
0809- AA        1210        TAX          WINDOW INDEX
080A- BD 36 08  1220        LDA WINDOW.DATA,X
080D- 85 22     1230        STA WNDTOP
```

RamWorks II®

The Best Selling, Most Compatible, Most Recommended, Most Expandable Card Available.



64K to 16 MEG! RamWorks II Is Number One.

It's simple, RamWorks II sells the most because it does the most.

The AppleWorks Amplifier.

While RamWorks II is recognized by all memory intensive programs, NO other expansion card comes close to offering the multitude of enhancements to AppleWorks that RamWorks II does. Naturally, you'd expect RamWorks II to expand the available desktop, after all Applied Engineering was a year ahead of everyone else *including Apple* in offering more than 55K in AppleWorks and we still provide the largest AppleWorks desktops available. But a larger desktop is just part of the story. Just look at all the AppleWorks enhancements that even Apple's own card does not provide and *only* RamWorks II does. With a 256K or larger RamWorks II, all of AppleWorks will automatically load itself into RAM dramatically increasing speed by eliminating all the time required to access the program disk drive. Now switch from word processing to spreadsheet to database at the speed of light with no wear on disk drives.

Only RamWorks II eliminates AppleWorks' internal memory limits, increasing the maximum number of records available from 1,350 to over 15,000. *Only* RamWorks II increases the number of lines permitted in the word processing mode from 2,250 to over 15,000. And *only* RamWorks II (256K or larger) offers a built-in printer buffer, so you won't have to wait for your printer to stop before returning to AppleWorks. RamWorks II even expands the clipboard. And auto segments large files so they can be saved on two or more disks.

RamWorks II, nothing comes close to enhancing AppleWorks so much.

The Most Friendly, Most Compatible Card Available.

Using RamWorks II couldn't be easier because it's compatible with more off-the-shelf software than any other RAM card. Popular programs like AppleWorks, Pinpoint, Catalyst, MouseDesk, HowardSoft, FlashCalc, The Spread Sheet, Managing Your Money, SuperCalc 3a, and MagiCalc to name a few (and *all* hardware add on's like ProFile and Sider hard disks). RamWorks II is even compatible with software written for Apple cards. But unlike other cards, RamWorks II plugs into the IIe auxiliary slot providing our super sharp 80 column text in a completely integrated system while leaving expansion slots 1 through 7 available for other peripheral cards.

Highest Memory Expansion.

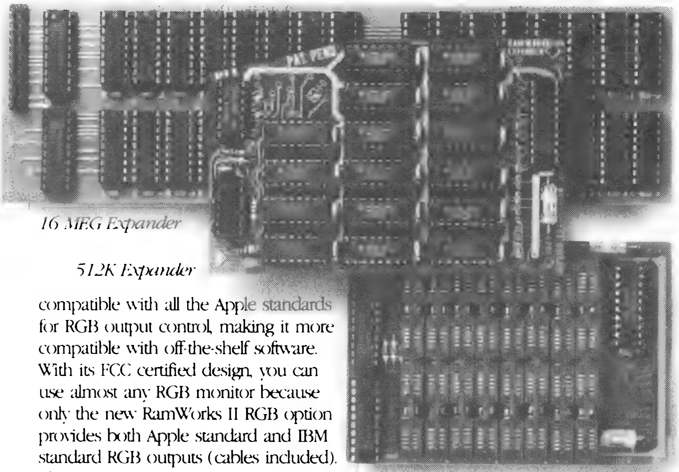
Applied Engineering has always offered the largest memory for the IIe and RamWorks II continues that tradition by expanding to 1 full MEG on the main card using standard RAMs, more than most will ever need (1 meg is about 500 pages of text)...but if you do ever need more, RamWorks II has the widest selection of expander cards available. Additional 512K, 2 MEG, or multiple 16 MEG cards just snap directly onto RamWorks II by plugging into the industry's only low profile (no slot 1 interference) fully decoded memory expansion connector. You can also choose non-volatile, power independent expanders allowing permanent storage for over 20 years.

It Even Corrects Mistakes.

If you've got some other RAM card that's not being recognized by your programs, and you want RamWorks II, you're in luck. Because all you have to do is plug the memory chips from your current card into the expansion sockets on RamWorks II to recapture most of your investment!

The Ultimate in RGB Color.

RGB color is an option on RamWorks II and with good reason. Some others combine RGB output with their memory cards, but that's unfair for those who don't need RGB *and* for those that do. Because if you don't need RGB Applied Engineering doesn't make you buy it, but if you want RGB output you're in for a nice surprise because the RamWorks II RGB option offers better color graphics plus a more readable 80 column text (that blows away any composite color monitor). For only $129 it can be added to RamWorks II, giving you a razor sharp, vivid brilliance that most claim is the best they have ever seen. You'll also appreciate the multiple text colors (others only have green) that come standard. But the RamWorks II RGB option is more than just the ultimate in color output because unlike others, it's fully



16 MEG Expander

512K Expander

2 Meg Expander

compatible with all the Apple standards for RGB output control, making it more compatible with off-the-shelf software. With its FCC certified design, you can use almost any RGB monitor because only the new RamWorks II RGB option provides both Apple standard and IBM standard RGB outputs (cables included). The RGB option plugs into the back of RamWorks II with no slot 1 interference (works on the original RamWorks, too) and remember you can order the RGB option with your RamWorks II or add it on at a later date.

True 65C816 16 Bit Power.

RamWorks II has a built-in 65C816 CPU port for direct connection to our optional 65C816 card. The only one capable of linearly addressing more than 1 meg of memory for power applications like running the Lotus 1-2-3™ compatible program, VIP Professional. Our 65C816 card does not use another slot but replaces the 65C02 yet maintains full 8 bit compatibility.

Endorsed by the Experts.

Steve Wozniak, creator of the Apple Computer said "I wanted a memory card for my Apple that was fast, easy to use, and very compatible; so I bought RamWorks." A+ magazine said "Applied Engineering's RamWorks is a boon to those who must use large files with AppleWorks...I like the product so much that I am buying one for my own system." inCider magazine said "RamWorks II is the most powerful auxiliary slot memory card available for your IIe, and I rate it four stars...For my money, Applied Engineering's RamWorks II is king of the hill."

Apple experts everywhere are impressed by RamWorks II's expandability, versatility, ease of use, and the sheer power and speed that it adds to any IIe. With a RamWorks II in your Apple, you'll make IBM PC's and AT's look like slowpokes.

It's Got It All

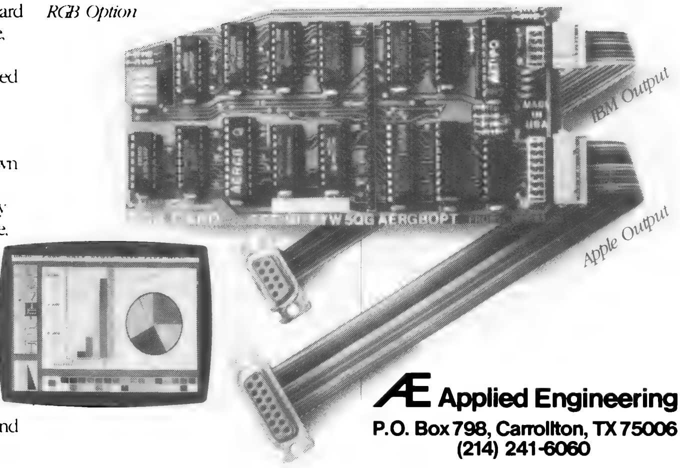
- 15 day money back guarantee
- 5 year hassle free warranty insures coverage no matter where you purchase
- Built-in super sharp 80 column display, (with or without RGB)
- Expandable to 1 MEG on main card
- Expandable to 16 meg with expander card, with NO slot 1 interference
- Can use 64K or 256K RAMs
- Powerful linear addressing 16 bit coprocessor port
- Automatic AppleWorks expansion up to 3017K desktop
- Accelerates AppleWorks
- Built-in AppleWorks printer buffer
- The only large RAM card that's 100% compatible with all IIe software
- RamDrive™ the ultimate disk emulation software included free
- Memory is easily partitioned allowing many programs to be in memory at once
- Compatible, RGB option featuring ultra high resolution color graphics and multiple text colors, with cables for both Apple and IBM type monitors
- Built-in self diagnostics software
- Lowest power consumption (patent pending)
- Takes only one slot (auxiliary) even when fully expanded
- Software industry standard
- Advanced Computer Aided Design
- Used by Apple Computer, Steve Wozniak and virtually all software companies
- Displays date and time on the AppleWorks screen with any PRO-DOS compatible clock
- Much, much more!

RamWorks II with 64K	$179
RamWorks II with 256K	$219
RamWorks II with 512K	$269
RamWorks II with 1 MEG	$369
RamWorks II with 1.5 MEG	$539
RamWorks II with 3 to 16 MEG	CALL
65C816 16 Bit Card	$159
RGB Option	$129
256K Upgrade	$ 50

RamWorks II. The industry standard for memory expansion of the Apple IIe.

ORDER YOUR RamWorks II TODAY. 9 a.m. to 11 p.m. 7 days, or send check or money order to Applied Engineering. MasterCard, Visa and C.O.D. welcome. Texas residents add 5⅛% sales tax. Add $10.00 if outside U.S.A.

RGB Option



Applied Engineering
P.O. Box 798, Carrollton, TX 75006
(214) 241-6060

```
080F- BD 37 08 1240        LDA WINDOW.DATA+1,X
0812- 85 23    1250        STA WNDBOT
0814- 20 58 FC 1260        JSR HOME      CLEAR THE WINDOW
               1270 *---DISPLAY MESSAGE, IF ANY------
0817- A0 00    1280        LDY #0
0819- 20 2D 08 1290 .1     JSR GET.NEXT.CALL.BYTE
081C- F0 08    1300        BEQ .2        END OF MESSAGE
081E- 09 80    1310        ORA #$80      ...JUST IN CASE
0820- 20 ED FD 1320        JSR COUT
0823- 4C 19 08 1330        JMP .1
               1340 *---RETURN TO CALLER-------------
0826- A5 01    1350 .2     LDA PNTR+1    HI BYTE
0828- 48       1360        PHA
0829- A5 00    1370        LDA PNTR      LO BYTE
082B- 48       1380        PHA
082C- 60       1390        RTS
               1400 *--------------------------------
               1410 GET.NEXT.CALL.BYTE
082D- E6 00    1420        INC PNTR      LO BYTE
082F- D0 02    1430        BNE .1
0831- E6 01    1440        INC PNTR+1    HI BYTE
0833- B1 00    1450 .1     LDA (PNTR),Y
0835- 60       1460        RTS
               1470 *--------------------------------
               1480 WINDOW.DATA
0836- 00 18 00
0839- 03 09 12
083C- 14 18    1490        .DA #0,#24,#0,#3,#9,#18,#20,#24
               1500 *--------------------------------
               1510 T
083E- 20 00 08 1520        JSR MSG.IN.WINDOW
0841- 02       1530        .DA #2        TOP WINDOW
0842- D4 CF D0
0845- A0 CC C9
0848- CE C5 A0
084B- CF C6 A0
084E- D4 C8 C5
0851- A0 D3 C3
0854- D2 C5 C5
0857- CE       1540        .AS -/TOP LINE OF THE SCREEN/
0858- 8D       1550        .HS 8D
0859- D3 C5 C3
085C- CF CE C4
085F- A0 CC C9
0862- CE C5 A0
0865- CF C6 A0
0868- D4 C8 C5
086B- A0 D3 C3
086E- D2 C5 C5
0871- CE       1560        .AS -/SECOND LINE OF THE SCREEN/
0872- 8A       1570        .HS 8A
0873- AE AE AE
0876- C1 CE C4
0879- A0 D4 C8
087C- C5 A0 D4
087F- C8 C9 D2
0882- C4       1580        .AS -/...AND THE THIRD/
0883- 00       1590        .HS 00        END MSG
0884- 20 C1 08 1600        JSR W
0887- 20 00 08 1610        JSR MSG.IN.WINDOW
088A- 06       1620        .DA #6        BOTTOM WINDOW
088B- CC C9 CE
088E- C5 A0 B2
0891- B1       1630        .AS -/LINE 21/
0892- 8A       1640        .HS 8A
0893- AE AE AE
0896- CC C9 CE
0899- C5 A0 B2
089C- B2       1650        .AS -/...LINE 22/
089D- 8A 8A    1660        .HS 8A.8A
089F- AE AE AE
08A2- C1 CE C4
08A5- A0 CC C9
08A8- CE C5 A0
08AB- B2 B4    1670        .AS -/...AND LINE 24/
08AD- 00       1680        .HS 00        END MSG
08AE- 20 C1 08 1690        JSR W
```

```
08B1- 20 00 08 1700        JSR MSG.IN.WINDOW
08B4- 00       1710        .DA #0       FULL SCREEN
08B5- CD D9 A0
08B8- CD C5 D3
08BB- D3 C1 C7
08BE- C5       1720        .AS -/MY MESSAGE/
08BF- 00       1730        .HS 00       END MSG
08C0- 60       1740        RTS
               1750 *--------------------------------
08C1- AD 00 C0 1760 W      LDA $C000    WAIT FOR KEY BEFORE CONTINUING
08C4- 10 FB    1770        BPL W
08C6- 8D 10 C0 1780        STA $C010
08C9- 60       1790        RTS
               1800 *--------------------------------
```

On Dividing a BCD Value by 4.............Bob Sander-Cederlof

The 6502 allows two kinds of addition and subtraction operations, depending on the state of the D-bit in the status register. After a SED (Set D) instruction, the ADC ana SBC instructions will operate in decimal mode; after CLD (CLear D), ADC and SBC will operate in binary mode.

In decimal mode the range of values in a byte is from $00 to $99. The left nybble is the ten's digit, and the right nybble is the unit's digit. The decimal mode makes some programs much easier to write, and others more difficult. Having both modes is nice.

In binary mode, if you want to divide by four you just shift the value right two bit-positions. If by 8, shift 3 times. And so on. In decimal mode, you can very easily divide by powers of ten; however, dividing by four is more difficult.

I needed a quick way to tell if a number in decimal mode was divisible by four. After inspecting the binary values of the decimal-mode numbers between 00 and 99 a, I found a way. If the ten's digit is even and the unit's digit 0, 4, or 8, the number is divisible by four. Also, if the ten's digit is odd and the unit's digit is 2 or 6, the number is divisible by four. This can be tested as follows:

```
       LDA VALUE
       AND #$13
       BEQ ...      ...TEN'S EVEN, UNITS=0,4,8
       EOR #$12
       BEQ ...      ...TEN'S ODD, UNITS=2,6
       ...          ...NOT DIVISIBLE
```

Next I needed a way to actually divide by four. Again I started by inspecting the various values involved. Simply shifting right twice does not do the job, except for numbers less than ten. You cannot even divide by two by simply shifting right once, unless the ten's digit is even. Hmmm.... If the ten's digit is odd, I could subtract 6 frist and then shift right once to divide by two. Doing all that twice woula result in a division by four. The subtraction must be done in binary mode, not decimal. The subroutine below in lines 1460-1590 will divide the decimal number in VALUE by four, truncating any remainder, and return the quotient in the A-register. Lines 1600-1700 show a shorter way to divide by two, provided you don't mind using the X-register.

Get TransWarp™. The fastest accelerator you can buy for your Apple™ IIe, II, or II+.

Computing at warp speed!

It's an experience you shouldn't miss. And with TransWarp, you won't have to. Because TransWarp will run your software up to 3.6 *times faster* — leaving other accelerators in the stardust!

No more yawning while your Apple™ slowly rearranges text or calculates spreadsheets. With 256K of ultra-fast RAM, TransWarp speeds up *all* Apple software — including AppleWorks, Supercalc 3a, Visicalc, and all educational software, graphics and games. And it's compatible with all standard peripheral cards (such as RamWorks II and Apple memory cards), Profile and Sider hard disks, 3½" UniDisks, 80-column cards, modems, clock cards, mouses and more! You name it, TransWarp accelerates it. There's even a 16 bit upgrade chip available should 16 bit software become available for the Apple.

"I recommend Applied Engineering products wholeheartedly."

*Steve Wozniak, the creator
of Apple Computer*

An important difference.

TransWarp's not the only speedup card on the market. But it's the only one that accelerates your Apple's main memory, ROM *and auxiliary memory*. And with more and more programs residing in auxiliary memory, buying anyone else's accelerator makes less and less sense. TransWarp even works with most D.M.A. devices including the Swyft™ card.

There's one more difference. Since TransWarp doesn't use memory caching, it accelerates *all* software — and not just most of it.

A cinch to use.

Simply plug TransWarp into any slot in your Apple II, II+ or IIe -- including slot 3 in the IIe. Instantly you'll be computing at speeds you only dreamed about before. And should you ever wish to run at normal speed, simply press the ESC key while turning your Apple on.

Since TransWarp is completely transparent, you won't need preboot disks or special software. It's ready to go right out of the package!

Speed = Productivity

Imagine the productivity gains you'll achieve when your programs are running over three times faster. TransWarp is so powerful, your Apple will make IBM PCs™ and even ATs™ look like slowpokes — whether you're planning taxes, plotting charts or playing games! Take a look at a few of the features that set TransWarp apart.

- 3.6 MHZ 65C02
- 256K of ultra-fast on-board RAM
- Accelerates main *and* auxiliary memory
- Low power consumption for cool operation
- Totally transparent operation with all software
- Plugs into any slot, including slot 3 on the Apple IIe
- Accelerated 16 bit option available

Satisfaction guaranteed!

Give *your* Apple the TransWarp advantage. With our risk-free 15-day money back guarantee, you have nothing to lose but wasted time. Call today!

TransWarp Accelerator	$279
16 bit upgrade (may add later) .	$89

For fast response:

Call Applied Engineering, 9 a.m. to 11 p.m., 7 days at (214) 241-6060. MasterCard, VISA and C.O.D. welcome. Texas residents add 5⅛% sales tax. Add $10.00 if outside U.S.A.

Or mail check or money order to Applied Engineering, P. O. Box 798, Carrollton, TX 75006.

Applied Engineering
The Apple enhancement experts.

P. O. Box 798, Carrollton, TX 75006 (214) 241-6060

To test my subroutines, I wrote some test programs. The first program, lines 1000-1370, runs through the values 00 to 99, printing ten values to a line. Each number that is evenly divisible by four is flagged with an asterisk. The secona program, lines 1720-1990, shows the quotient after calling DIVIDE.BCD.VALUE.BY FOUR.

I am sure there must be lots of other neat tricks possible by combining binary and decimal modes in the 6502. Do you know some? Send them in, and we will publish the best!

```
                  1000 *SAVE BCD MAGIC
                  1010 *--------------------------------
FD8E-             1020 CROUT  .EQ $FD8E
FDDA-             1030 PRBYTE .EQ $FDDA
FDED-             1040 COUT   .EQ $FDED
                  1050 *--------------------------------
00-               1060 VALUE  .EQ 0
                  1070 *--------------------------------
                  1080 T
0800- A9 00       1090        LDA #0        FOR VALUE = 0 TO $FF
0802- 85 00       1100 .1     STA VALUE
0804- A9 A0       1110        LDA #" "
0806- 20 ED FD    1120        JSR COUT
0809- A5 00       1130        LDA VALUE
080B- 20 DA FD    1140        JSR PRBYTE
                  1150 *--------------------------------
080E- 20 35 08    1160        JSR IS.BCD.VALUE.DIVISIBLE.BY.FOUR
0811- F0 03       1170        BEQ .2        ...YES
0813- A9 A0       1180        LDA #" "      ...NO
0815- 2C          1190        .HS 2C
0816- A9 AA       1200 .2     LDA #"*"
0818- 20 ED FD    1210        JSR COUT
                  1220 *--------------------------------
081B- A9 A0       1230        LDA #" "      SEPARATE ITEMS IN CHART
081D- 20 ED FD    1240        JSR COUT
0820- A5 00       1250        LDA VALUE     NEW LINE AFTER TEN VALUES
0822- 29 0F       1260        AND #$0F
0824- C9 09       1270        CMP #9
0826- D0 03       1280        BNE .3
0828- 20 8E FD    1290        JSR CROUT
                  1300 *---NEXT VALUE-------------------
082B- F8          1310 .3     SED           MUST DO ARITHMETIC
082C- A5 00       1320        LDA VALUE        IN DECIMAL MODE
082E- 18          1330        CLC
082F- 69 01       1340        ADC #1
0831- D8          1350        CLD           BACK TO BINARY
0832- 90 CE       1360        BCC .1        ...UNTIL WRAP-AROUND
0834- 60          1370        RTS
                  1380 *--------------------------------
                  1390 IS.BCD.VALUE.DIVISIBLE.BY.FOUR
0835- A5 00       1400        LDA VALUE     RETURN .EQ. STATUS IF YES
0837- 29 13       1410        AND #$13             .NE. STATUS IF NOT
0839- F0 02       1420        BEQ .1
083B- 49 12       1430        EOR #$12
083D- 60          1440 .1     RTS
                  1450 *--------------------------------
                  1460 DIVIDE.BCD.VALUE.BY.FOUR
083E- A5 00       1470        LDA VALUE
0840- 20 43 08    1480        JSR DIVIDE.BCD.VALUE.BY.TWO
                  1490 DIVIDE.BCD.VALUE.BY.TWO
0843- 48          1500        PHA
0844- 29 10       1510        AND #$10
0846- F0 05       1520        BEQ .1
0848- 68          1530        PLA
0849- E9 06       1540        SBC #6
084B- 4A          1550        LSR
084C- 60          1560        RTS
084D- 68          1570 .1     PLA
084E- 4A          1580        LSR
084F- 60          1590        RTS
                  1600 *--------------------------------
                  1610 SHORTER.DIV.BY.TWO
0850- 4A          1620        LSR
0851- AA          1630        TAX
0852- 29 08       1640        AND #8
0854- F0 03       1650        BEQ .1
0856- CA          1660        DEX
0857- CA          1670        DEX
0858- CA          1680        DEX
0859- 8A          1690 .1     TXA
085A- 60          1700        RTS
                  1710 *--------------------------------
                  1720 D
085B- A9 00       1730        LDA #0        FOR VALUE = 0 TO $FF
085D- 85 00       1740 .1     STA VALUE
085F- A9 A0       1750        LDA #" "
0861- 20 ED FD    1760        JSR COUT
0864- A5 00       1770        LDA VALUE
0866- 20 DA FD    1780        JSR PRBYTE
0869- A9 AE       1790        LDA #"."
086B- 20 ED FD    1800        JSR COUT
                  1810 *--------------------------------
086E- 20 3E 08    1820        JSR DIVIDE.BCD.VALUE.BY.FOUR
0871- 20 DA FD    1830        JSR PRBYTE
                  1840 *--------------------------------
0874- A9 A0       1850        LDA #" "      SEPARATE ITEMS IN CHART
0876- 20 ED FD    1860        JSR COUT
0879- A5 00       1870        LDA VALUE     NEW LINE AFTER TEN VALUES
087B- 29 0F       1880        AND #$0F
087D- C9 09       1890        CMP #9
087F- D0 03       1900        BNE .3
0881- 20 8E FD    1910        JSR CROUT
                  1920 *---NEXT VALUE-------------------
0884- F8          1930 .3     SED           MUST DO ARITHMETIC
0885- A5 00       1940        LDA VALUE        IN DECIMAL MODE
0887- 18          1950        CLC
0888- 69 01       1960        ADC #1
088A- D8          1970        CLD           BACK TO BINARY
088B- 90 D0       1980        BCC .1        ...UNTIL WRAP-AROUND
088D- 60          1990        RTS
```

Booting into 80 Columns.........................Bill Morgan

The ProDOS version of the S-C Macro Assembler is carefully written to operate in either 40 or 80 columns. When you boot the disk the assembler starts out in the 40 column mode, because we couldn't take for granted that you would have (or want) the 80 column display. Well it turns out that most people (myself included) are using 80 columns and are getting tired of typing PR#3 every time they start up the assembler.

Marc Wolfgram called up today from Wisconsin to ask how to make the assembler start up in 80 columns, and that finally got me around to finding out how. It's embarassingly easy: just a two-byte patch. Here's the procedure, assuming you're in S-C Macro Assembler ProDOS:

```
UNLOCK SCASM.SYSTEM
BLOAD SCASM.SYSTEM,A$2000,TSYS
$6001:00 C3
BSAVE SCASM.SYSTEM,A$2000,TSYS,L17920
LOCK SCASM.SYSTEM
```

We just changed the IO.INIT call from JMP MON.HOME to JMP $C300, and that's all there is to it! Now the next time you boot up, the assembler will be in 80 column mode. RESET will return you to 40 columns. PR#3 or NEW will restore 80 columns.

Thanks, Marc, for prompting me to find out about this.

Faster Boot and More Space for DOS 3.3....Bob Sander-Cederlof

A freshly initialized DOS 3.3 disk has 496 free sectors, less whatever is used by your HELLO program. There are 16 more sectors that are either never used or which are wasted, in tracks 0 and 2. The following program modifies the code which writes the DOS image and the code which reads it back during boot, so that the entire image fits in tracks 0 and 1. A further change makes the space in track 2 available for normal files.

The new boot procedure actually is faster than the standard one, and all the new code takes less space than that which is replaced. All you give up is the ability to boot into machines with less than 48K. Does anyone still have one?

Standard DOS 3.3 stores the DOS image in two pieces. The code destined for $B600-BFFF is on track 0, sectors 0 through 9. The code for $9D00-B500 follows, from track 0/sector 10 through track 2/sector 4. Sectors 5-15 of track 2 are not used. The information stored in sectors 3 and 4 of track 2 (aimed at $B400-B5FF) is useless, because all this space is variables for DOS which do not need to be initialized. The same goes for sector 5 of track 0. The contents of sectors 10 and 11 of track 0 is not used on a "slave" disk, which is what you get with the INIT command. My disks have to stay slave disks, because we are going to reshuffle everything around so all the unused sectors end up in track 2.

My new layout stores $9D00-9DFF in track 0/sector 5, and $9E00-B3FF in track 0/sector 10 through track 1/sector 15. The following table summarizes the old and new layouts.

Sector	Track 0		Track 1		Track 2	
	Old	New	Old	New	Old	New
0	B6	B6	A1	A4	B1	..
1	B7	B7	A2	A5	B2	..
2	B8	B8	A3	A6	B3	..
3	B9	B9	A4	A7	B4	..
4	BA	BA	A5	A8	B5	..
5	BB	9D	A6	A9	..	..
6	BC	BC	A7	AA	..	..
7	BD	BD	A8	AB	..	..
8	BE	BE	A9	AC	..	..
9	BF	BF	AA	AD	..	..
10	..	9E	AB	AE	..	..
11	..	9F	AC	AF	..	..
12	9D	A0	AD	B0	..	..
13	9E	A1	AE	B1	..	..
14	9F	A2	AF	B2	..	..
15	A0	A3	B0	B3	..	..

I published the complete commented disassembly of the code which writes the DOS image on a disk and the code for the second stage boot in AAL way back in October, 1981. The second stage boot code begins at $B700, and the DOS writer starts at $B74A. They both use a subroutine at $B793 to read/write a range of sectors. I preserved the starting points for these

DISASM 2.2e - AN INTELLIGENT DISASSEMBLER : $30.00

Investigate the inner workings of machine language programs. DISASM converts machine code into meaningful, symbolic source. Creates a standard text file compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even lets you substitute your own meaningful labels. (100 commonly used Monitor and Pg Zero names included.) An address-based triple cross reference table is provided to screen or printer. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his new ASSEMBLY COOKBOOK. For entire Apple II family including the new Apple //c (with all the new opcodes). SOURCE CODE available for an additional $30.00

LOW LOW PRICE !!! C-PRINT For The APPLE //c : $69.00

Connect standard parallel printers to an Apple //c. C-PRINT is a hardware accessary that plugs into the standard Apple //c printer serial port. The other end plugs into any printer having a standard 36 pin centronics-type parallel connector. Just plug in and print! High speed data transfer at 9600 Baud. No need to reconfigure serial port or load software drivers for text printing.

FONT DOWNLOADER & EDITOR : $39.00

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. All special printer functions (like expanded, compressed etc.) apply to custom fonts. Full HIRES screen editor lets you create your own characters and special graphics symbols. Compatible with many parallel printer I/F cards. User driver option provided. For Apple II, II+, //e. Specify printer: Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/100, or OkiData 92/93.

The Font Downloader & Editor for the Apple Imagewriter Printer. For use with Apple II, II+, //e (with SuperSerial card) and the new Apple //c (with builtin serial interface).

FONT LIBRARY DISKETTE #1 : $19.00 Contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

The 'PERFORMER' CARD : $39.00

Plugs into any slot to convert a 'dumb' centronics-type printer I/F card into a 'smart' one. Command menu eliminates need to remember complicated ESC codes. Features include perforation skip, auto page numbering with date & title. Includes large HIRES graphics & text screen dumps. Specify printer: MX-80 with Graftrax-80, MX-100, MX-80/100 with Graftraxplus, NEC 8092A, C.Itoh 8510 (Prowriter), OkiData 82A/83A with Okigraph & OkiData 92/93. SOURCE CODE: $30.00

FIRMWARE FOR APPLE-CAT: The 'MIRROR' ROM : $25.00

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support. Uses superset of Apple's Comm card and Micromodem II commands. SOURCE CODE: $50.00

RAM/ROM DEVELOPMENT BOARD : $30.00

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into $CnOO-CnFF and $C800-CFFF.

ALL NEW !!! MIDI *MUSIC PRODUCTS*

MIDI means Musical Instrument Digital Interface. Use your computer with any MIDI-equipped music keyboard for entertainment and music education. Low cost MIDI player interface cable, complete with 6 song demo disk: $49.00. Thousands of popular songs available soon on diskette (also compatible with Passport MIDI interface). Products for both the Apple IIc and Commodore 64/128. Unique general purpose MIDI expander cable and gender changer also available. Send SASE for product descriptions and prices.

Avoid a $3.00 handling charge by enclosing full payment with order. VISA/MC and COD phone orders accepted.

RAK-WARE 41 Ralph Road W. Orange N J 07052 (201) 325-1885 two routines in the program which follows, but there is a lot of new empty space. If you are interested, you could go ahead and shove all the code segments together, patch all the calls for the new locations, and get one big area of free space for adding new features.

I was able to save coding space in several ways. First, by deciding that I would not worry about running in less than 48K. Second, that I could eliminate the extra code used to clobber the language card. This is a very common patch anyway, because most of us do not want to have to keep re-loading the language card area just because we re-boot DOS. Third, by eliminating the redundant calls to $FE89 and $FE93. The first stage boot does both of these just before jumping to the second stage boot, so there is no reason to do them again. And fourth, by being more efficient. If you want to, you can save even more by doing away with the subroutine at $B7C2: part of it is redundant, and the rest can be combined with the code at $B74A.

The standard DOS boot first loads $B600-BFFF from track 0, and then skips out to track 2 to read the rest hind-end-first. The track steps are 0-1-2-1-0. My new version starts in track 0, reads it all, then reads all of track 1, and it is done. The track steps are simply 0-1. It is a lot faster. However, the overall boot time is not significantly faster, due to the time spent finding track 0 in the first place, and the time spent loading the HELLO program.

Lines 1060-1140 install the new code. The entire $B7 page is replaced, as well as a single byte at $AEB3. This byte changes the VTOC on the newly initialized disk so that track 2 is available. While I was looking at this area, I noticed that the VTOC written on the new disk is not necessarily correct. DOS does not create an entirely new VTOC for the new disk. The bitmap area is new, and several other bytes are set up. However, DOS does not store any values in the bytes which tell how many tracks, sectors per track, sector size, and T/S entries per T/S list. This means that if the last access to a disk prior to initializing a new one was to a non-standard disk, the VTOC may be incorrect on the new disk. If I load a file from a large volume on my Sider, and then INIT a floppy, the floppy's VTOC indicates 32 sectors per track and 50 tracks! Ouch! Beware!

Lines 1180-1480 are the second stage boot code. The first stage boot is located at $B601, and actually executed at $801. It loads in sectors 0-9 of track 0 into $B600-BFFF, calls $FE89 and $FE93 to set the standard 40-column input hooks, and then jumps to $B700 with the slot*16 in the X-register. My stage two begins by copying the information which came from sector 5, now found at $BB00-BBFF, to the place it belongs at $9D00-9DFF (lines 1270-1320). Next I set up a call to my RWFT subroutine.

RWFT stands for Read/Write From Table. I have a table that describes all of the segments which must be loaded from the disk during boot, or written during initialization. Stage two boot must read the same things written by initialization, but init-ing requires first writing the stuff which will be loaded by stage one boot. Stage two boot calls RWFT with A=1 (read opcode for the IOB) and Y=2 (skipping the first two entries in the table). Initialization calls RWFT with A=2 (write opcode) and Y=0 (start at the beginning of the table).

RWFT gets four items out of the table for each step. The page number and sector number indicate the end of the range to be read or written. The count tells how many pages (or sectors) need to be read or written. All of the sectors must be in the track specified by the table entry. After one range has been read, RWFT steps to the next. The table terminates when the page address of 0 is found.

For some reason the code at $AEFF looks like this:

```
       AEFF-  JSR $B7C2
       AF02-  JSR $B74A
```

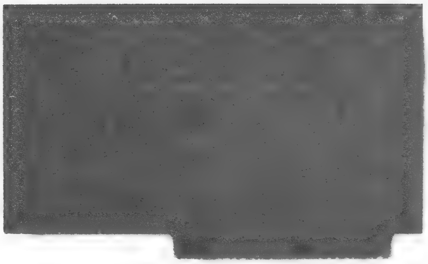
Both of these subroutines are never called from any other place, so they could be combined into one. Doing so would save several bytes. Furthermore, at least with my new RWFT program, lines 2120 and 2130 could be deleted, saving six more bytes.

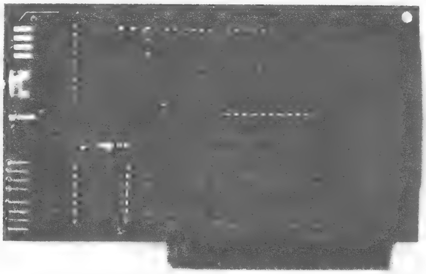
There are still more ways to increase the storage on standard floppies, as you probably know. You can shorten the catalog, make a few other patches, and use some sectors in track 17 ($11).

You can usually use more than 35 tracks, since most drives will handle at least 36 and many a full 40. This also only takes a few simple patches. At $AEB5 you normally find a value $8C. Add 4 to this value for each additional track. This controls the loop that builds the bitmap of available sectors in the VTOC. The byte at $BEFE controls how many tracks the formatter in RWTS lays down. It is normally $23 (decimal 35), so add one for each additional track. Just before you start the INIT command, change the byte at $B3EF. This is normally $23, the number of tracks. Add 1 for each additional track. You have to be sure to do this last patch just prior to the INIT, because reading or writing another disk will cause it to be changed back.

Incidentally, this reminds me of the potential bug I mentioned above regarding writing out an incorrect VTOC. Once today I tried to catalog a disk that had been only partially initialized. The tracks had been written, but no VTOC or catalog sectors were. Of course I got an I/O ERROR. Next I decided to INIT that same disk. It went through the formatting stage, then bombed out with an I/O error when trying to write the catalog. Looking at the VTOC on this disk, the bytes for number of tracks, et cetera, were all zero!.

Now back to extra tracks. After making a disk with the extra tracks, you really need to check them to be sure your drive handles them. Use a disk zap program and try to write on the last track. Then try to write on the previous track. If your drive will go out that far, you will be successful. If you get an error trying to find the next to the last track, keep





backing up until you find a track that does work. All the ones in between were written in the same location on the disk surface as the last track. If there were any missing tracks, you need to reformat the disk with fewer tracks.

And interesting side not to this discussion is that you could format a disk with LESS than 35 tracks if you wish. Just so you at least include track 17 ($11), you can reduce the values at $BEFE, $B3EF, and $AEB5 and stop short of a full disk. Some copy protection schemes do this, along with other tricks, to frustrate the making of copies.

```
                 1000 *SAVE S.B700-B7FF DOS 3.3
                 1010 *--------------------------------
B5BC-            1020 FMP.SUBCOD .EQ $B5BC
B5F9-            1030 FMW.VOLUME .EQ $B5F9
BD00-            1040 RWTS       .EQ $BD00
                 1050 *--------------------------------
                 1060 INSTALL
0800- A0 00      1070        LDY #0        COPY NEW CODE INTO DOS
0802- B9 11 08   1080 .1     LDA NEW.B700,Y     $B700...B7FF
0805- 99 00 B7   1090        STA $B700,Y
0808- C8         1100        INY
0809- D0 F7      1110        BNE .1
080B- A9 08      1120        LDA #8        PATCH TO INCLUDE TRACK 2
080D- 8D B3 AE   1130        STA $AEB3     AS FREE SPACE
0810- 60         1140        RTS
                 1150 *--------------------------------
                 1160 NEW.B700  .PH $B700
                 1170 *--------------------------------
                 1180 BOOT.STAGE2
B700- 8E E9 B7   1190        STX IOB.SLOT16
B703- 8E F7 B7   1200        STX IOB.PRVSLT
B706- 8A         1210        TXA           SLOT*16
B707- 4A         1220        LSR           GET SLOT #
B708- 4A         1230        LSR
B709- 4A         1240        LSR
B70A- 4A         1250        LSR
B70B- AA         1260        TAX           X = SLOT NUMBER
                 1270 *---COPY BB00-FF TO 9D00-FF------
B70C- A0 00      1280        LDY #0
B70E- B9 00 BB   1290 .1     LDA $BB00,Y
B711- 99 00 9D   1300        STA $9D00,Y
B714- 88         1310        DEY
B715- D0 F7      1320        BNE .1
                 1330 *---SET CURRENT TRACKS @ 0-------
B717- 98         1340        TYA           A = Y = 0
B718- 9D F8 04   1350        STA $4F8,X
B71B- 9D 78 04   1360        STA $478,X
                 1370 *---BUILD RWFT CALL--------------
B71E- C8         1380        INY           Y = 1
B71F- 8C F8 B7   1390        STY IOB.PRVDRV
B722- 8C EA B7   1400        STY IOB.DRIVE      DRIVE = 1
B725- 98         1410        TYA                A = 1 (READ OPCODE)
B726- C8         1420        INY                Y = 1 (RWFT INDEX)
B727- 20 4E B7   1430        JSR RWFT
                 1440 *---COLD START DOS---------------
B72A- A2 FF      1450        LDX #$FF
B72C- 9A         1460        TXS           EMPTY STACK
B72D- 8E EB B7   1470        STX IOB.VOLUME
B730- 4C 84 9D   1480        JMP $9D84     DOS HARD ENTRY
                 1490 *--------------------------------
B733-            1500        .BS $B74A-*        <<<FILLER>>>
                 1510 *--------------------------------
                 1520 *      WRITE DOS IMAGE ON TRACKS 0-2
                 1530 *--------------------------------
                 1540 WRITE.DOS.IMAGE
B74A- A9 02      1550        LDA #2        WRITE OPCODE FOR RWTS
B74C- A0 00      1560        LDY #0        RWFT INDEX
```

```
                     1570 *--------------------------------
                     1580 *   READ/WRITE FROM TABLE
                     1590 *--------------------------------
                     1600 RWFT
B74E- 8D F4 B7  1610        STA IOB.OPCODE
B751- 8C 8A B7  1620 .1     STY RWFT.INDEX
B754- B9 8B B7  1630        LDA RWFT.ADDR,Y
B757- F0 2F     1640        BEQ .3        ...END OF RWFT TABLE
B759- 8D F1 B7  1650        STA IOB.BUFFER+1
B75C- B9 90 B7  1660        LDA RWFT.TRACK,Y
B75F- 8D EC B7  1670        STA IOB.TRACK
B762- B9 94 B7  1680        LDA RWFT.SECTOR,Y
B765- 8D ED B7  1690        STA IOB.SECTOR
B768- B9 98 B7  1700        LDA RWFT.COUNT,Y
B76B- 8D 89 B7  1710        STA RWFT.N
B76E- A9 B7     1720 .2     LDA /IOB
B770- A0 E8     1730        LDY #IOB
B772- 20 B5 B7  1740        JSR ENTER.RWTS
B775- B0 F7     1750        BCS .2        ...TRY AGAIN IF ERROR
B777- CE ED B7  1760        DEC IOB.SECTOR    NEXT SECTOR
B77A- CE F1 B7  1770        DEC IOB.BUFFER+1  NEXT PAGE
B77D- CE 89 B7  1780        DEC RWFT.N
B780- D0 EC     1790        BNE .2
B782- AC 8A B7  1800        LDY RWFT.INDEX
B785- C8        1810        INY
B786- D0 C9     1820        BNE .1        ...ALWAYS
B788- 60        1830 .3     RTS
                1840 *--------------------------------
B789-           1850 RWFT.N     .BS 1
B78A-           1860 RWFT.INDEX .BS 1
                1870 *--------------------------------
B78B- BF 9D A3
B78E- B3 00     1880 RWFT.ADDR   .HS BF.9D.A3.B3.00
B790- 00 00 00
B793- 01        1890 RWFT.TRACK  .HS 00.00.00.01
B794- 09 05 0F
B797- 0F        1900 RWFT.SECTOR .HS 09.05.0F.0F
B798- 0A 01 06
B79B- 10        1910 RWFT.COUNT  .HS 0A.01.06.10
                1920 *--------------------------------
B79C-           1930        .BS $B7B5-*       <<<FILLER>>>
                1940 *--------------------------------
                1950 *      ENTER RWTS
                1960 *--------------------------------
                1970 ENTER.RWTS
B7B5- 08        1980        PHP           SAVE STATUS ON STACK
B7B6- 78        1990        SEI           DISABLE INTERRUPTS
B7B7- 20 00 BD  2000        JSR RWTS      CALL RWTS
B7BA- B0 03     2010        BCS .1        ERROR RETURN
B7BC- 28        2020        PLP           RESTORE STATUS
B7BD- 18        2030        CLC           SIGNAL NO RWTS ERROR
B7BE- 60        2040        RTS           RETURN TO CALLER
B7BF- 28        2050 .1     PLP           RESTORE STATUS
B7C0- 38        2060        SEC           SIGNAL RWTS ERROR
B7C1- 60        2070        RTS           RETURN TO CALLER
                2080 *--------------------------------
                2090 *      SET UP RWTS TO WRITE DOS
                2100 *--------------------------------
                2110 SETUP.WRITE.DOS
B7C2- AD BC B5  2120        LDA FMP.SUBCOD  IMAGE ADDRESS
B7C5- 8D F1 B7  2130        STA IOB.BUFFER+1
B7C8- A9 00     2140        LDA #0
B7CA- 8D F0 B7  2150        STA IOB.BUFFER
B7CD- AD F9 B5  2160        LDA FMW.VOLUME  VOLUME #
B7D0- 49 FF     2170        EOR #$FF        UNCOMPLEMENT IT
B7D2- 8D EB B7  2180        STA IOB.VOLUME
B7D5- 60        2190        RTS
                2200 *--------------------------------
                2210 *      CLEAR 256 BYTES STARTING AT ($42,43)
                2220 *--------------------------------
                2230 ZERO.CURRENT.BUFFER
B7D6- A9 00     2240        LDA #0
B7D8- A8        2250        TAY
B7D9- 91 42     2260 .1     STA ($42),Y
B7DB- C8        2270        INY
B7DC- D0 FB     2280        BNE .1
B7DE- 60        2290        RTS
```

```
                 2300 *-------------------------------
B7DF-            2310        .BS $B7E8-*      <<<FILLER>>>
                 2320 *-------------------------------
                 2330 *     IOB FOR RWTS CALLS
                 2340 *-------------------------------
                 2350 IOB
B7E8- 01         2360 IOB.TYPE    .HS 01     0--MUST BE $01
B7E9- 60         2370 IOB.SLOT16  .HS 60     1--SLOT # TIMES 16
B7EA- 01         2380 IOB.DRIVE   .HS 01     2--DRIVE # (1 OR 2)
B7EB- 00         2390 IOB.VOLUME  .HS 00     3--DESIRED VOL # (0 MATCHES ANY)
B7EC-            2400 IOB.TRACK   .BS 1      4--TRACK # (0 TO 34)
B7ED-            2410 IOB.SECTOR  .BS 1      5--SECTOR # (0 TO 15)
B7EE- FB B7      2420 IOB.PNTDCT  .DA DCT    6--ADDRESS OF DCT
B7F0-            2430 IOB.BUFFER  .BS 2      8--ADDRESS OF DATA
B7F2- 00 01      2440 IOB.SECTSZ  .DA 256   10--# BYTES IN A SECTOR
B7F4-            2450 IOB.OPCODE  .BS 1     12--0=SEEK, 1=READ, 2=WRITE, OR 4=FORMAT
B7F5-            2460 IOB.ERROR   .BS 1     13--ERROR CODE: 0, 8, 10, 20, 40, 80
B7F6-            2470 IOB.ACTVOL  .BS 1     14--ACTUAL VOLUME # FOUND
B7F7- 60         2480 IOB.PRVSLT  .HS 60    15--PREVIOUS SLOT #
B7F8- 01         2490 IOB.PRVDRV  .HS 01    16--PREVIOUS DRIVE #
B7F9-            2500             .BS 2
B7FB- 00 01 EF
B7FE- D8         2510 DCT      .HS 0001EFD8
B7FF-            2520          .BS 1
                 2530 *-------------------------------
```

A "Gotcha!" in New //c ROMs.................Robert H. Bernard

Apple seems to have installed a bug in the new ROM for the Apple //c which affects DOS 3.3. I am talking about the 3.5 ROM that supports Unidisk 3.5 and AppleTalk.

The new bug manifests itself when you use the control-IxxN command to either serial port. The older //c ROMs accumulated the "xx" number in $47F; the new ones do it in $47E. Location $47E is supposed to be dedicated to slot 6, the slot where the disk drives are. DOS uses $47E to keep track of the current track position for drive 1. So, after doing the serial port command to set line length, the next time DOS tries to look at drive 1 it will have to re-calibrate.

Re-calibration is not a disaster, but it is annoying. A needless and not noiseless waste of time. To avoia it with the new ROMs, you have to save and restore the contents of $47E around any serial port command that involves scanning a numeric field.

I have looked through the entire listing of the 3.5 ROM that came with my upgrade kit, and there does not appear to be any reason why this variable was moved. Location $47F is not used for any new value that I can see.

Even though the Apple //c Technical Reference Manual reserves $47E for the firmware, and ProDOS doesn't use the cell, using a "slot 6" screen-hole for a slot 1 and 2 activity is a serious breach of the protocol for their use that dates back to the earliest Wozdays. I know Apple is dropping (or at least decreasing) their support of DOS 3.3, but this is ridiculous!

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage). A subscription to the newsletter and the Monthly Disk containing all source code is $45 per year in the US, Canada and Mexico, and $87 to other countries.